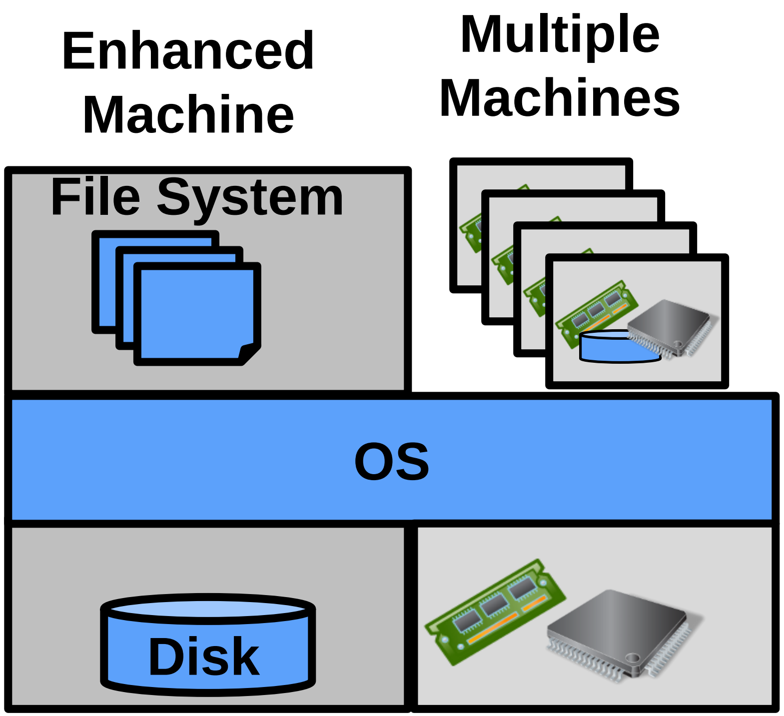
Temporal Protection in Real-Time Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

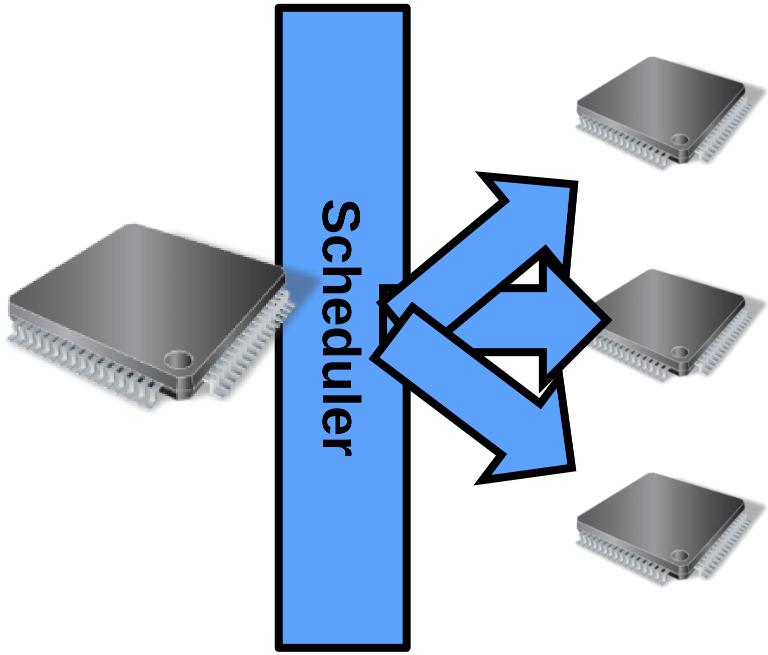
November 2016

OS Dual Objective



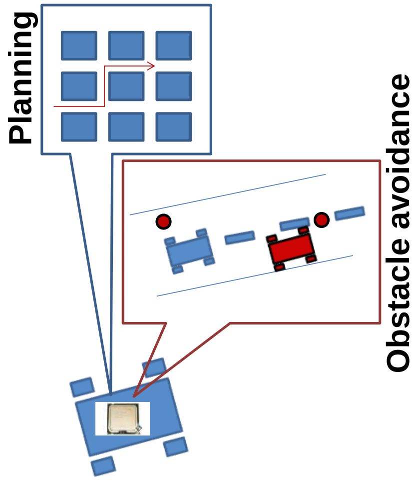
Icons credit: http://www.doublejdesign.co.uk

Time-Sharing CPU – Round robin

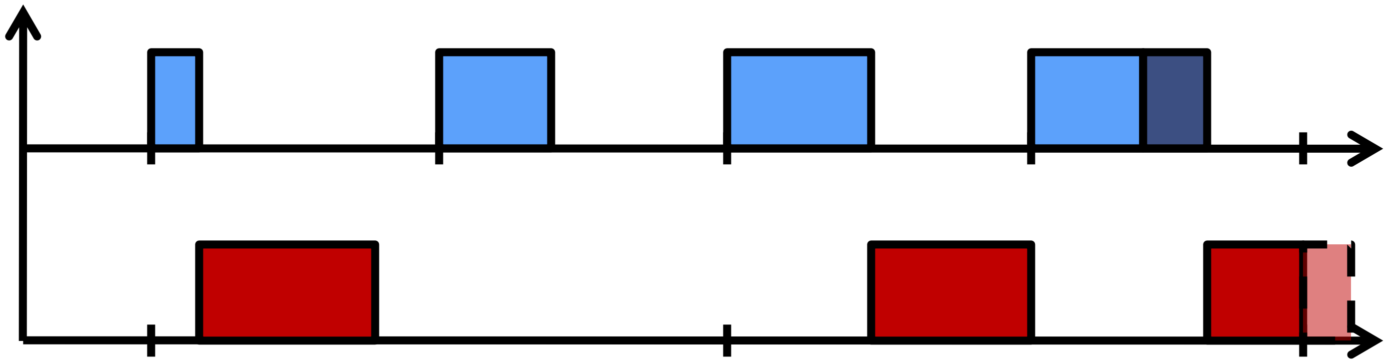


Same time requirement – Fair Scheduling

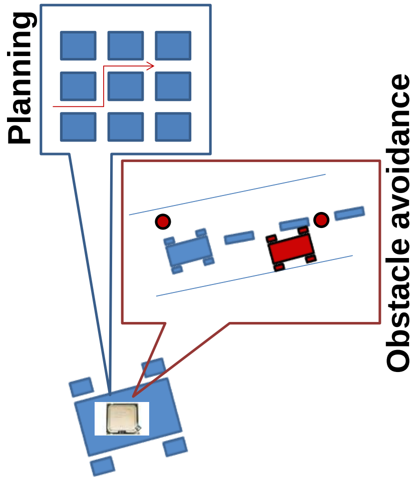
Consolidation of Mixed-Criticality Tasks



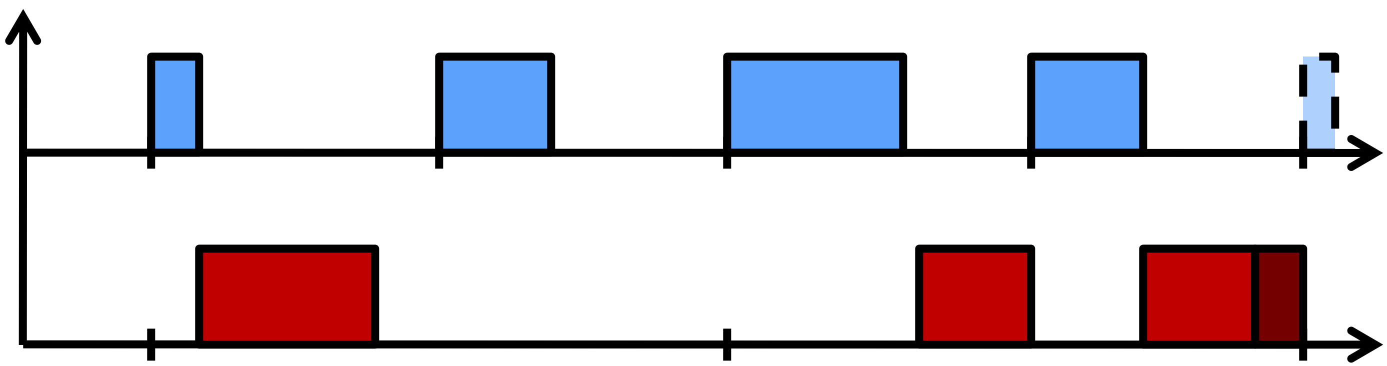
Shared Hardware
Can lead to cycle stealing



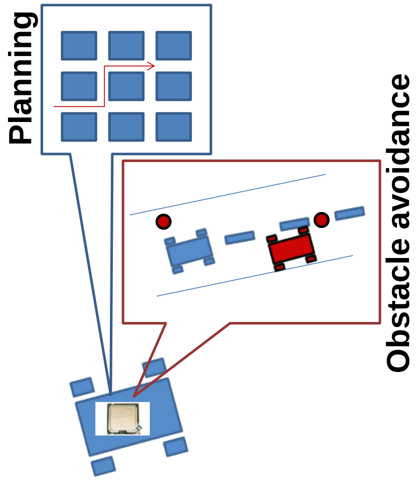
Consolidation of Mixed-Criticality Tasks



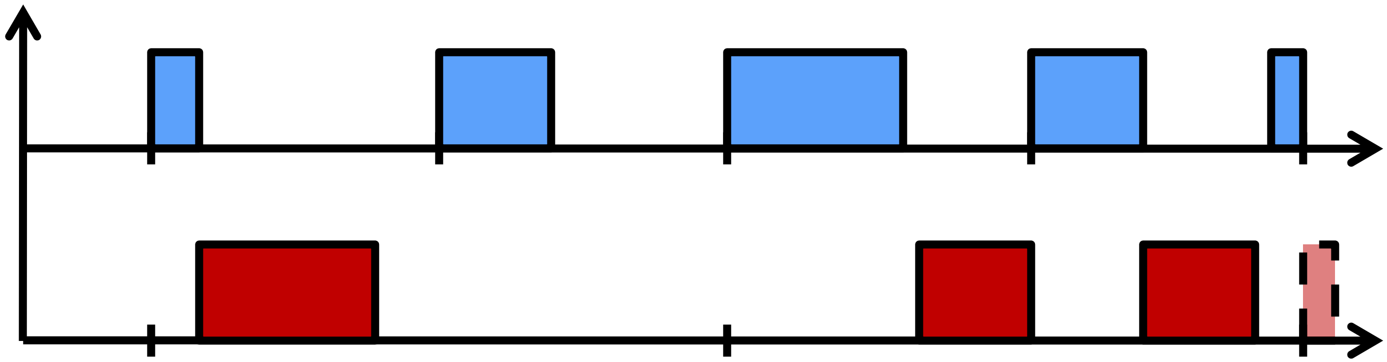
To avoid interference add temporal protection



Consolidation of Mixed-Criticality Tasks



BUT
Symmetric protection
leads to *criticality inversion*



Criticality Inversion

A higher-criticality task waits for a lower-criticality task to release a resource

- Symmetric temporal protection
- Scheduling policy is aimed at maximizing utilization (RMS/DMS/EDF)

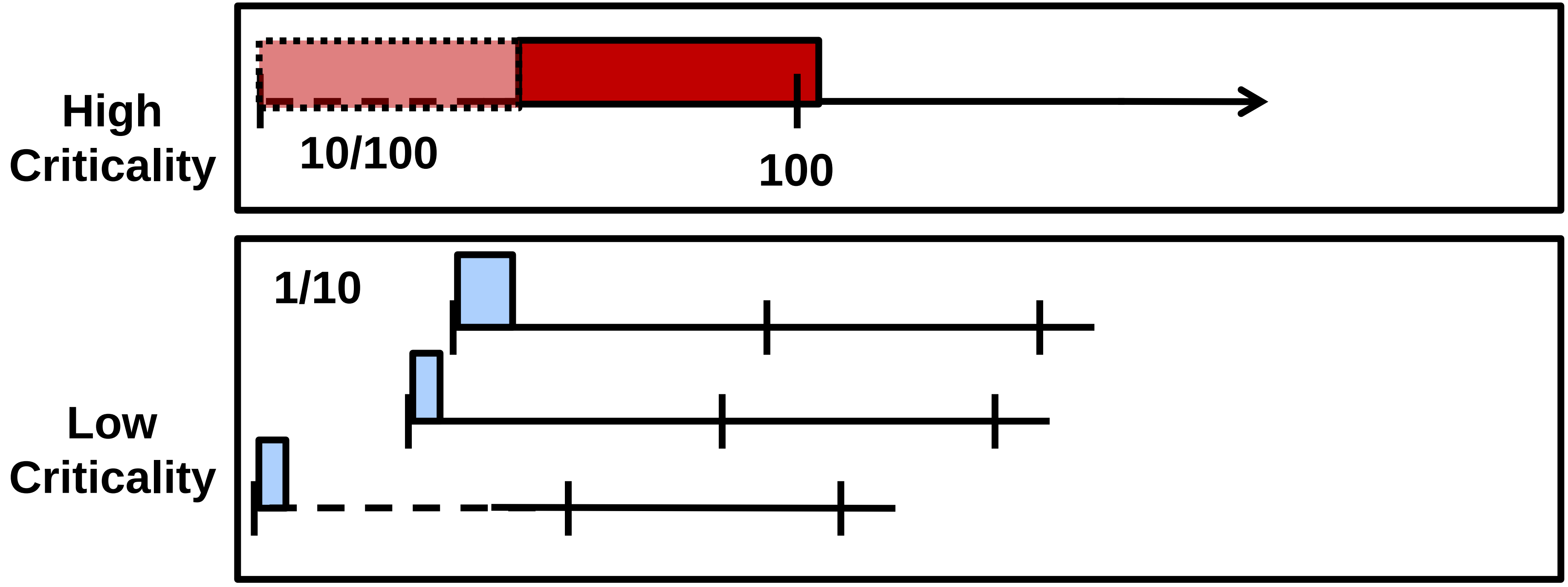
Rate-Monotonic Priority

Shorter Period → Higher Priority

- Ideal utilization

BUT: Poor Criticality Protection Due to Criticality Inversion

- If criticality order is opposite to rate-monotonic priority order



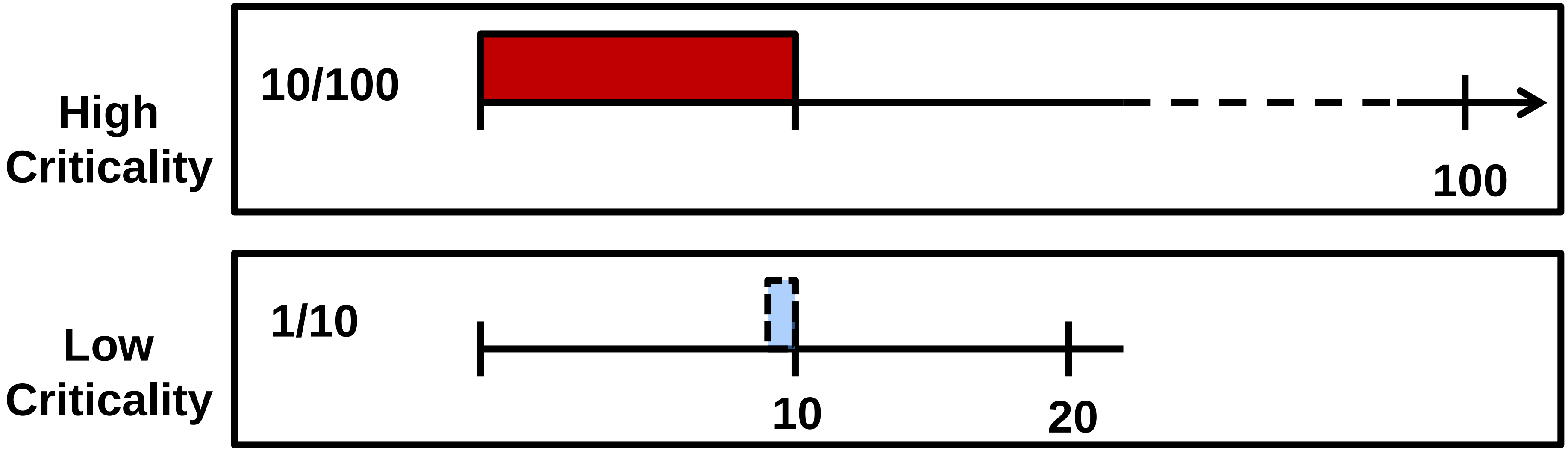
Criticality As Priority Assignment (CAPA)

Higher Criticality → Higher Priority

- Ideal criticality protection:
  - lower criticality cannot interfere with higher criticality

BUT: Poor Utilization Due to Priority Inversion

- If criticality order is opposite to rate-monotonic priority order



Task Model

$$\tau_i = \left(C_i, C_i^o, T_i, D_i, \zeta_i\right)$$

C_i **Normal Execution Budget of task *i***

C_i^o **Overload Execution Budget of task *i***

T_i **Period of task *i***

D_i **Deadline of task *i*** $D_i \leq T_i$

ζ_i **Criticality of task *i***

Zero-Slack Scheduling

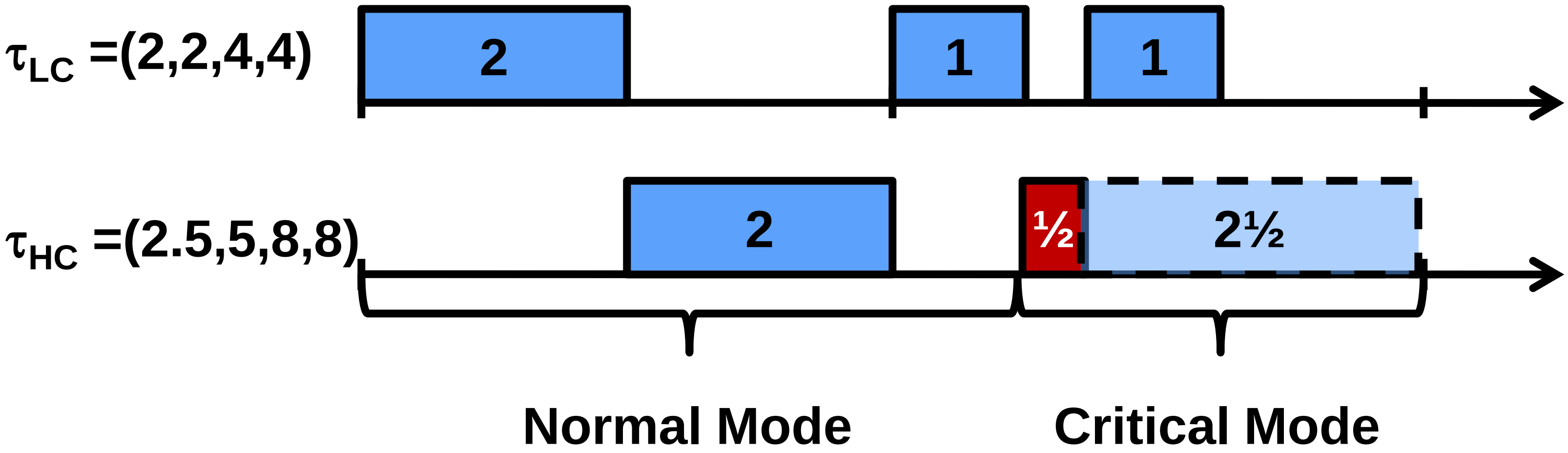
Start with RM

Calculate the last instant before τ_{HC} misses its deadline

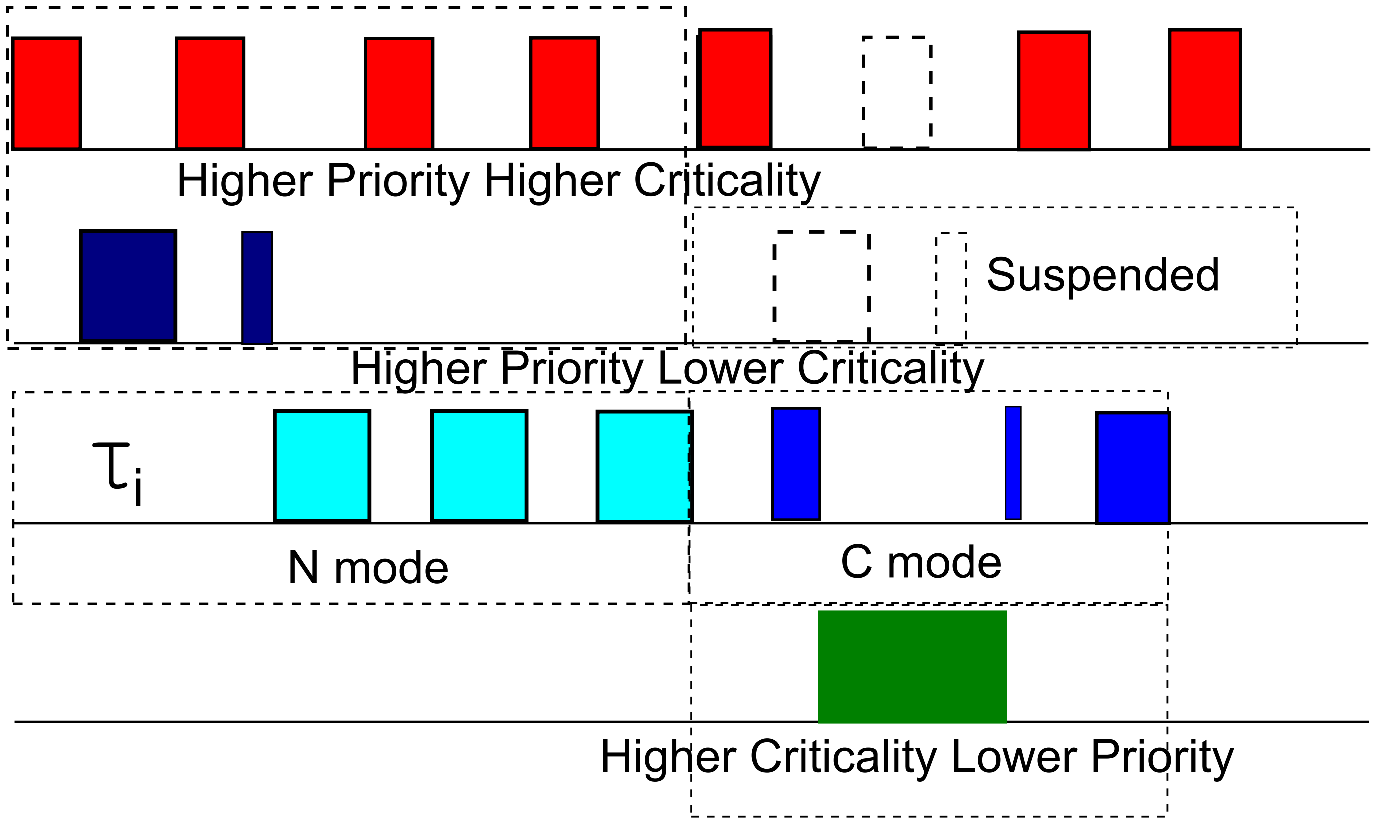
- this is called the ***zero-slack*** instant

Switch to criticality-as-priority

- Splits the execution window into
  - Normal mode (RM)
  - Critical mode (CAPA)



Critical Instant of a Task τ_i



Interference in Zero-Slack Scheduling

Task Set Divided into

- H^{lc} : Higher priority, lower criticality
- H^{hc} : Higher priority, higher criticality
- L^{lc} : Lower priority, lower criticality
- L^{hc} : Lower priority, higher criticality

Interfering tasks in normal mode (Normal mode)

- $H^{lc} + H^{hc} + L^{hc}$

Interfering tasks in critical mode (C mode)

- $H^{hc} + L^{hc}$

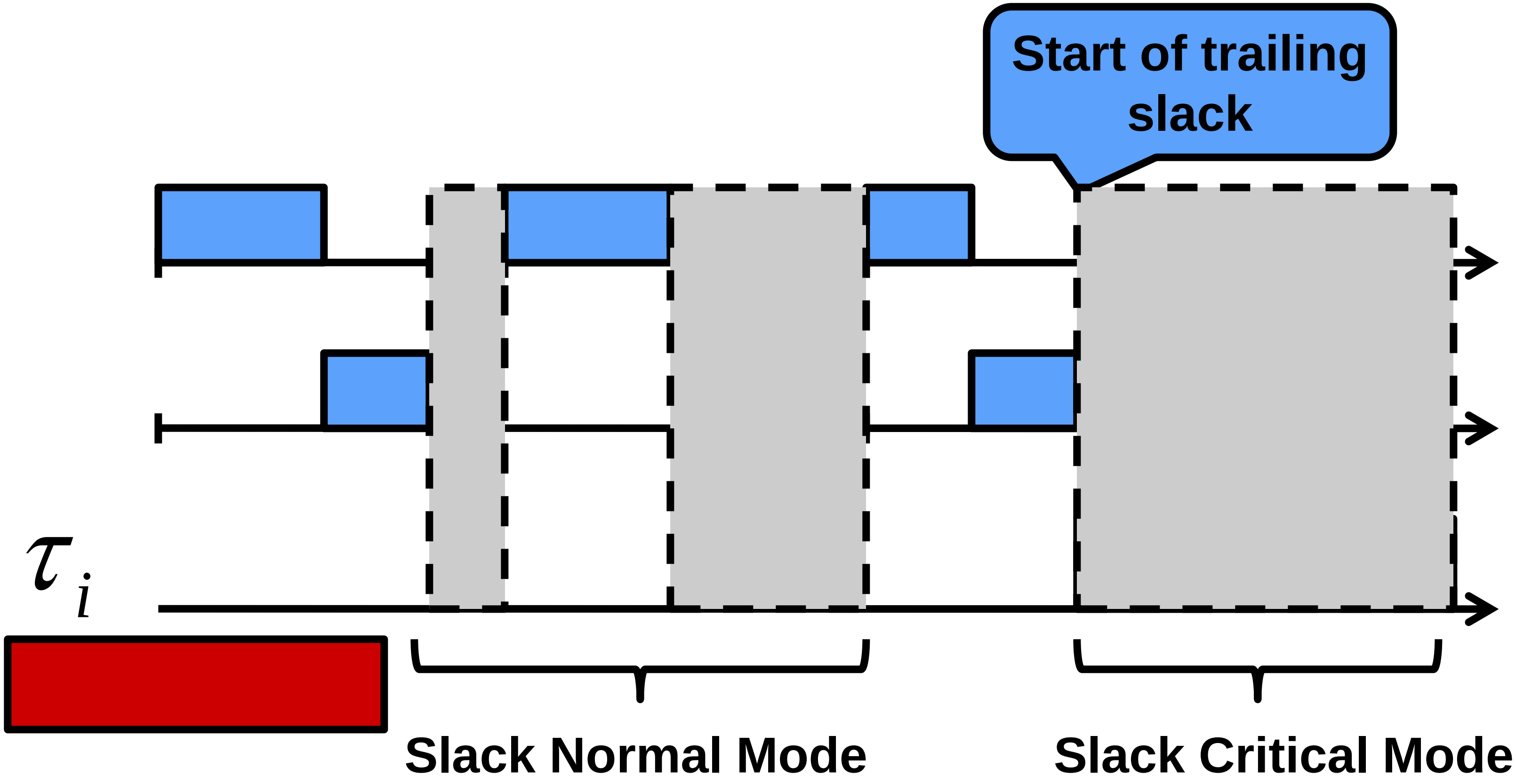
Scheduling Guarantee

A task τ_i is guaranteed C_i^o before D_i

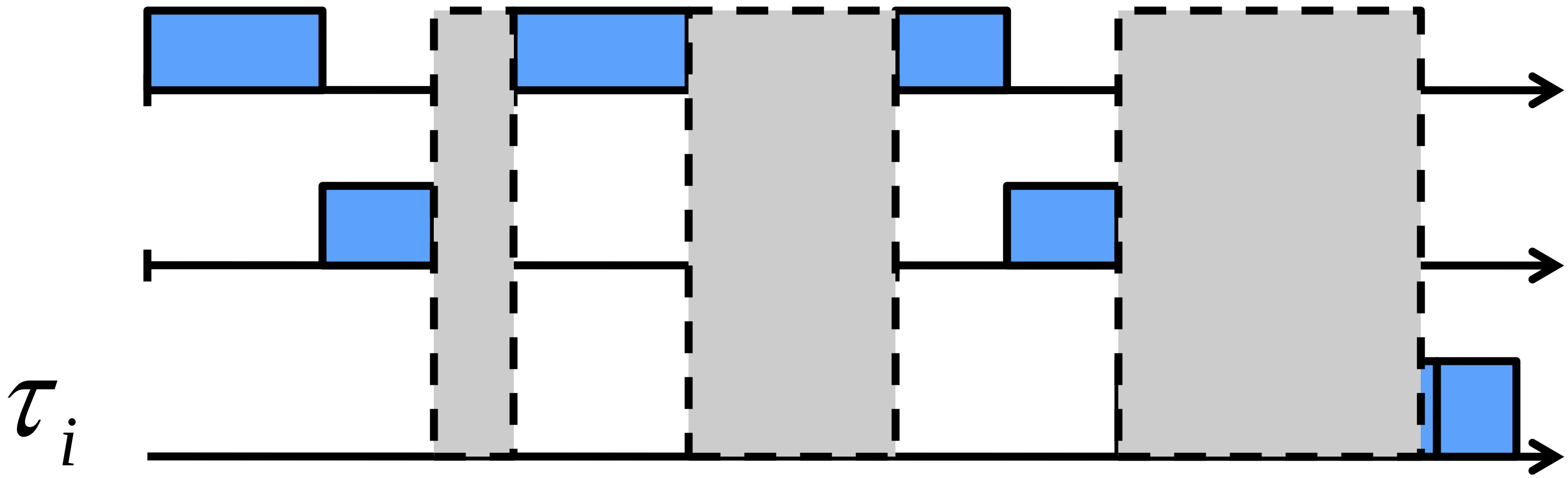
if no $\tau_j \mid \zeta_j < \zeta_i$

executes beyond its C_j

Calculating The Zero-Slack Instant



Calculating The Zero-Slack Instant



New slack can open after each iteration

Needs to repeat until no new slack opens

ZSRM Properties

Subsumes RM

- If criticalities are aligned to priorities
- No critical mode

Subsumes CAPA

- If not enough slack, only critical mode

Graceful Degradation

- In overloads, deadlines are missed in reverse criticality order

Implementation

ZSRM

Scheduling algorithm calculates zero-slack instants offline

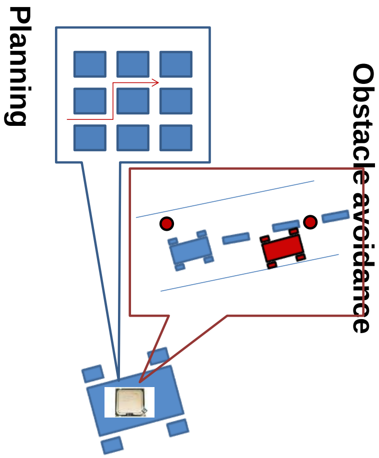
Linux/ RK

- Resource reservation in Linux
  - CPU, Net, Mem, Disk
- Bundled into resource sets that provide a form of virtual machine
- Multiple implementations
  - Nano/RK for sensor networks

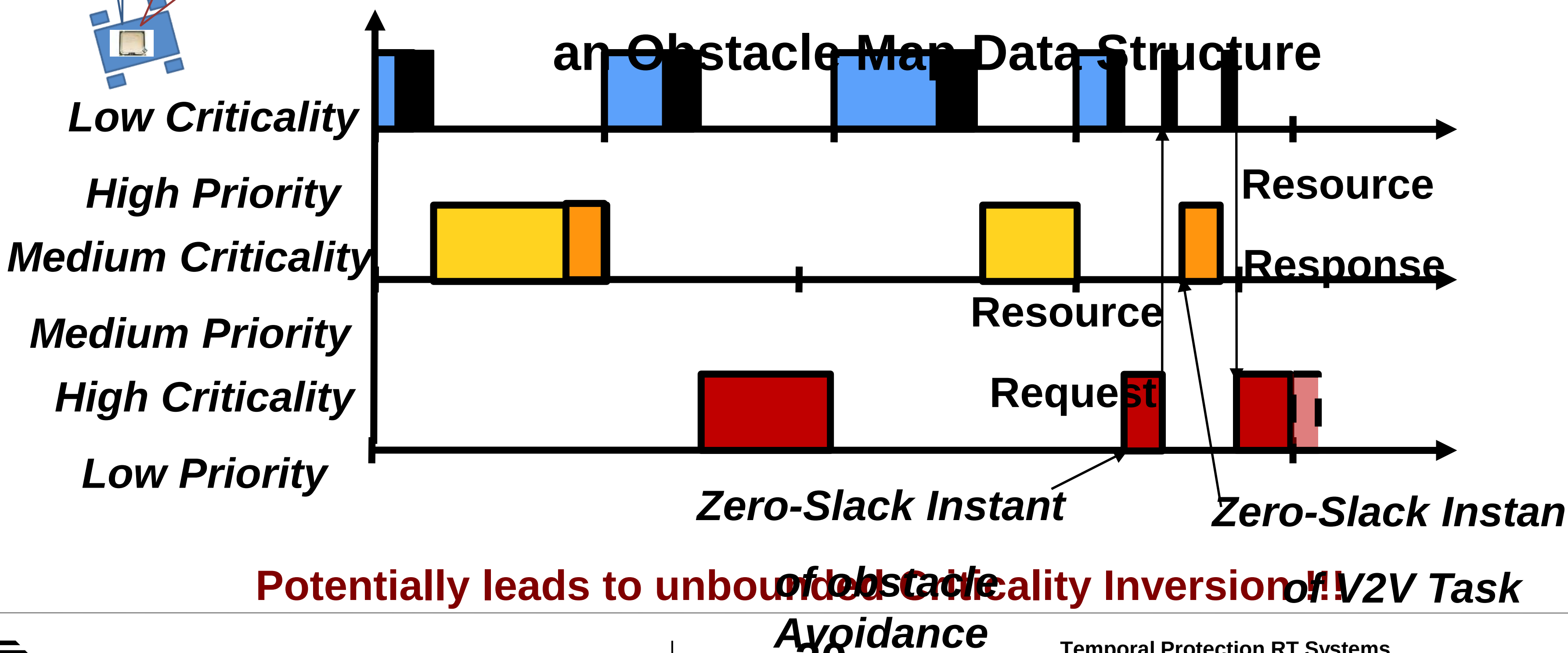
Special Zero-Slack Reserves

- Switch to critical mode
  - Stop lower-criticality tasks on zero-slack instant
- Tasks in critical mode in stack

What about Shared Resources?

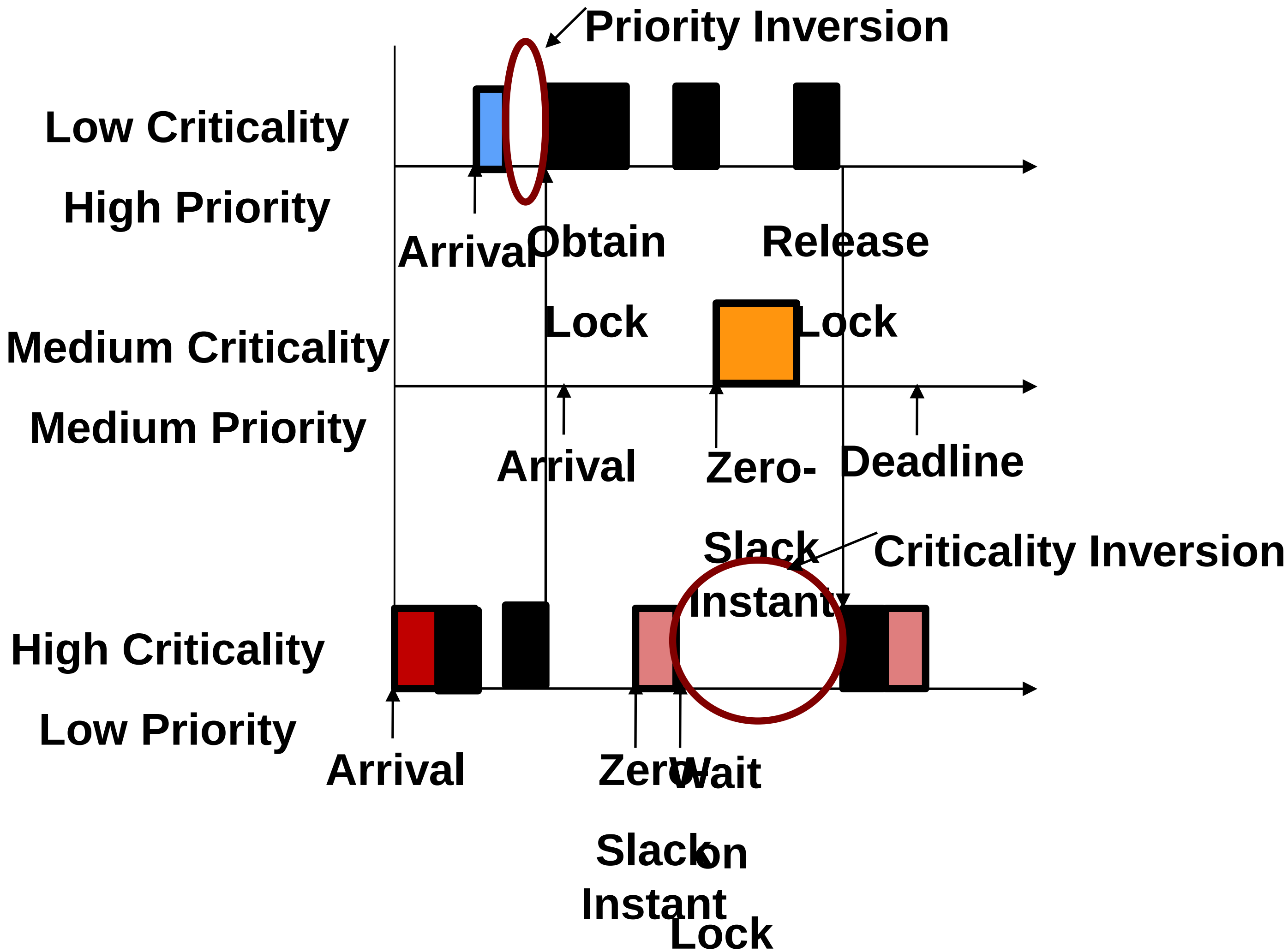


- Consider a new medium priority and Medium criticality task (say v2v task)
- Let Planning and Obstacle Avoidance share an Obstacle Map Data Structure



Potentially leads to unbounded Criticality Inversion!!!

Priority and Criticality Inversion



Blocking in Zero-Slack Scheduling

A job J_h waiting for a job J_l to exit critical section $Z_{l,k}$ is considered to be *blocked* at time t, *if and only if* one of the following conditions is satisfied at t:

1) The priority of J_l is lower than J_h's priority and J_l is running in its ***normal*** mode.

2) The criticality of J_l is lower than J_h's criticality and J_h is running in its ***critical*** mode.

Priority and Criticality Inheritance Protocol (PCIP)

PCIP Definition

A task τ_i that holds a lock to a resource can inherit the priority from a task τ_j and the criticality from a task τ_k (τ_k can be the same as τ_j), both requesting a lock to the resource held by τ_i as follows:

- τ_i *inherits the priority* of τ_j if τ_j 's priority is higher.
  - This inherited priority has an immediate effect on the scheduling of τ_i
- τ_i *inherits the criticality* of τ_k if τ_k's criticality is higher.
  - This criticality is used by τ_i immediately as soon as τ_k requests the lock held by τ_i

PCIP Possible Blocking

Consider a Job J_0

- $L_i^{hc}(J_0)$ ***is the set of jobs with***

lower priority and higher criticality

- $L_i^{lc}(J_0)$ ***is the set of jobs with***

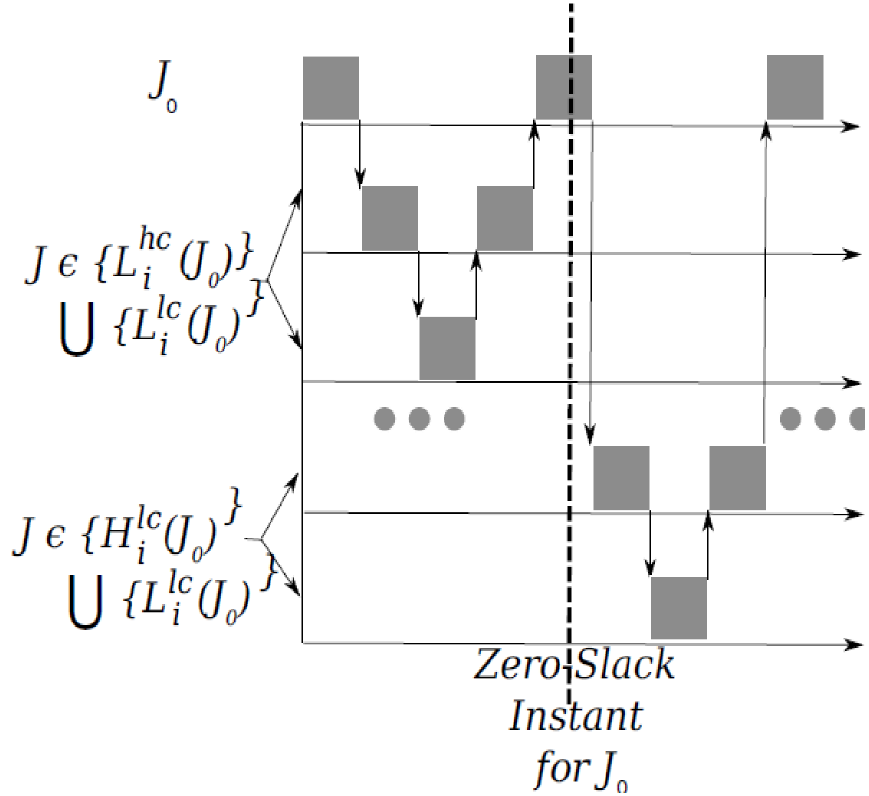
lower priority and lower criticality

- $H_i^{lc}(J_0)$ ***is the set of jobs with***

higher priority and lower criticality

Then $H_i^{hc}(J_0)$ all other sets can lead to priority or criticality inversion

$H_i^{hc}(J_0)$ ***is the set of jobs with***



PCIP Properties

- Under PCIP, given a job J_0 for which there are n jobs $\{J_1, J_2, \ldots, J_n\}$, with J_i in $\{ L_i^{hc}(J_0) \cup L_i^{lc}(J_0) \cup H_i^{lc}(J_0) \}$, job J_0 can be blocked for at most the duration of one critical section in each of $\beta^*_{0,i}$.
  - where,

    $\beta^*_{0,i}$ is the set of critical sections of J_i that can block J_0

- Under PCIP, if there are *"m"* locks which can block job *J,* then *J* can be blocked at most *"m"* times in its *normal* mode and blocked at most *"m"* times in its *critical* mode.

PCIP Illustration

Low Criticality $(P_1 \, V_1)$

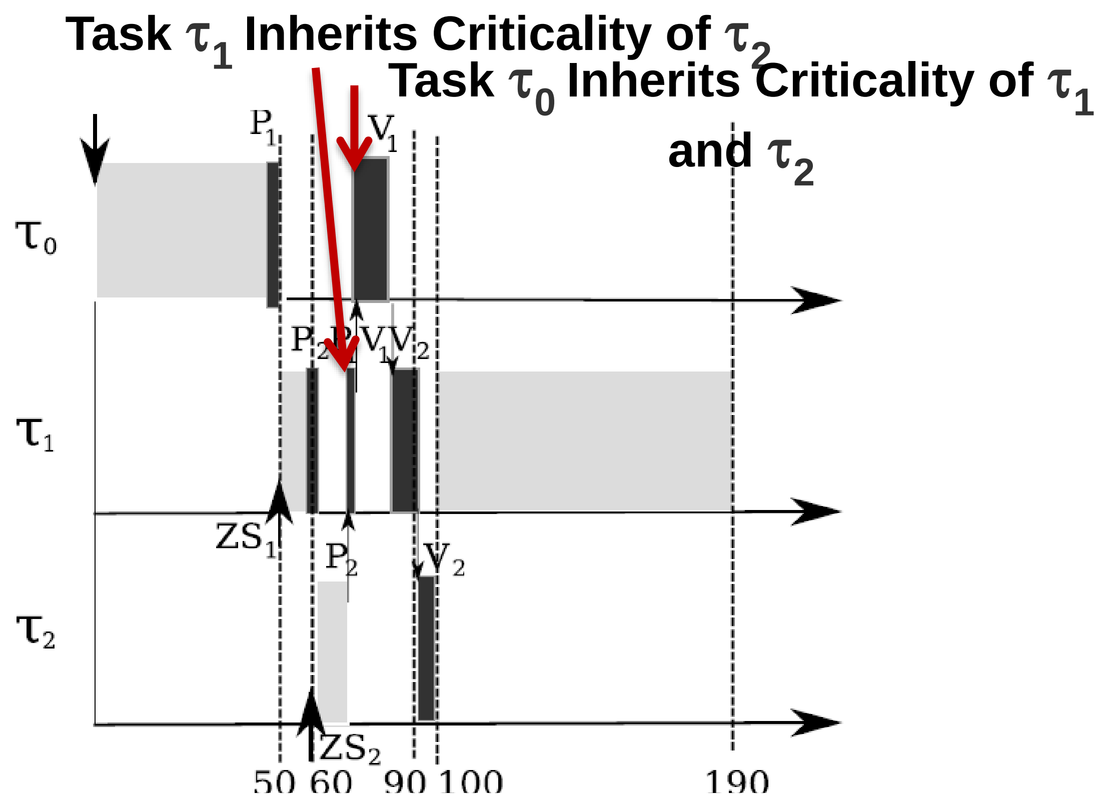
High Priority

Medium Criticality $(P_1 \, P_2 \, V_2 \, V_1)$

Medium Priority

High Criticality $(P_2 \, V_2)$

Low Priority



Task	C	C^o	T	D	Criticality	Priority
τ_0	10	70	100	100	2	0
τ_1	20	100	200	200	1	1
τ_2	40	200	400	400	0	2

Priority and Criticality Ceiling Protocol (PCCP)

PCCP Definition

- Each lock is assigned both a *priority ceiling* and a *criticality ceiling*
  - Priority ceiling is the highest possible priority of any locker of the lock
  - Criticality ceiling is the highest possible criticality of any locker of the lock
- Both the *priority ceiling* and the *criticality ceiling* of a lock are acquired by task whenever it holds the lock

PCCP – Maximum Blocking

- Each job *J* can only be blocked twice
  - At most once in ***Normal*** execution mode
  - At most once in ***Critical*** execution mode

- Each job J_w can block job *J* only once
  - Otherwise, J_w is $L_i^{lc}(J_0)$
  - And, Job *J* has to be blocked by J_w once in Normal mode
  - However, J_w cannot obtain the processor again as it is in $L_i^{lc}(J_0)$!!!

PCCP - No Deadlocks

- Under PCCP, no job J_k can preempt another job J_i while J_i holds a lock (i.e. is inside the critical section) that is also accessed by J_k.
- PCCP prevents *Transitive Blocking*
- PCCP *prevents* *Deadlocks*

PCCP Illustration

Task τ_0 acquires the Priority and Criticality Ceiling of Lock (P_1, V_1)

Task τ_1 acquires the Priority and Criticality Ceiling of Lock (P_2, V_2)

Low Criticality (P_1 V_1)

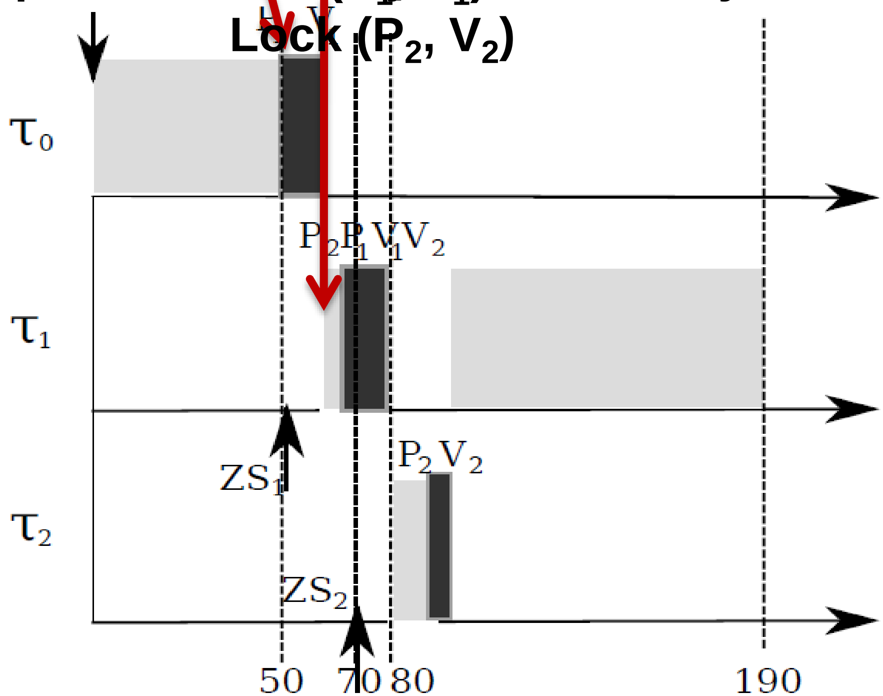
High Priority

Medium Criticality (P_1 P_2 V_2 V_1)

Medium Priority

High Criticality (P_2 V_2)

Low Priority



Task	C	C^o	T	D	Criticality	Priority
τ_0	10	70	100	100	2	0
τ_1	20	100	200	200	1	1
τ_2	40	200	400	400	0	2

PCIP Blocking Term Analysis

- PCIP Blocking Term B_i for Task τ_i

$$B_i = \min\Big(\sum_{\tau_j \in \{H_i^{lc} \cup L_i^{lc} \cup L_i^{hc}\}} \lambda(\beta^*_{i,j}), \sum_{\Psi_{i,k} \in \Psi_i} 2\Lambda(\Psi_{i,k})\Big)$$

where,

- $\beta^*_{i,j}$ is the set of critical sections of τ_j that can block τ_i
- $\lambda(\beta^*_{i,j})$ is the length of the *longest critical sections* of $\beta^*_{i,j}$ that can block task τ_i
- $\Lambda(\Psi_{i,j})$ is the length of the critical section protected by lock $\Psi_{i,j}$

PCCP Blocking Term Analysis

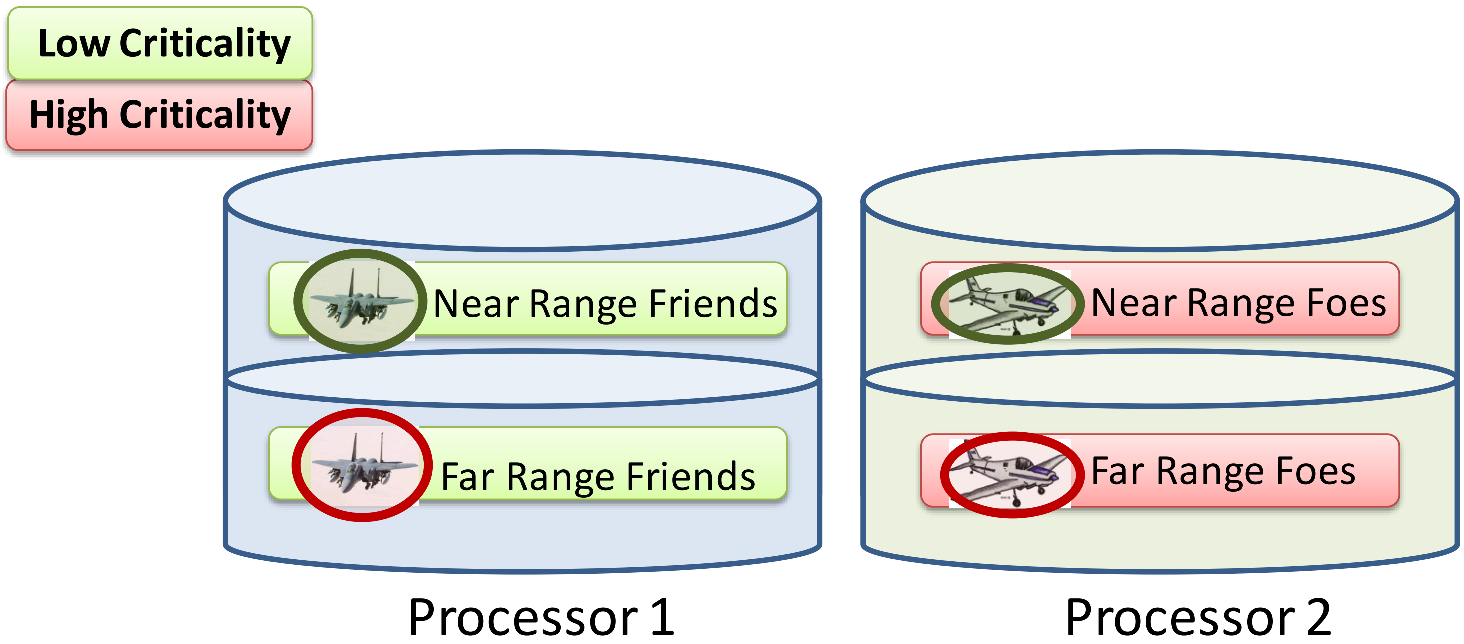
- PCCP Blocking Term B_i for Task τ_i

$$B_i = \max_{\tau_j \in \{H_i^{lc} \cup L_i^{lc} \cup L_i^{hc}\}} 2\lambda(\beta^*_{i,j})$$

where,

- $\beta^*_{i,j}$ is the set of critical sections of τ_j that can block task τ_i
- $\lambda(\beta^*_{i,j})$ is the length of the longest critical sections of $\beta^*_{i,j}$ that can block task τ_i

Criticality isolation strategy (S)

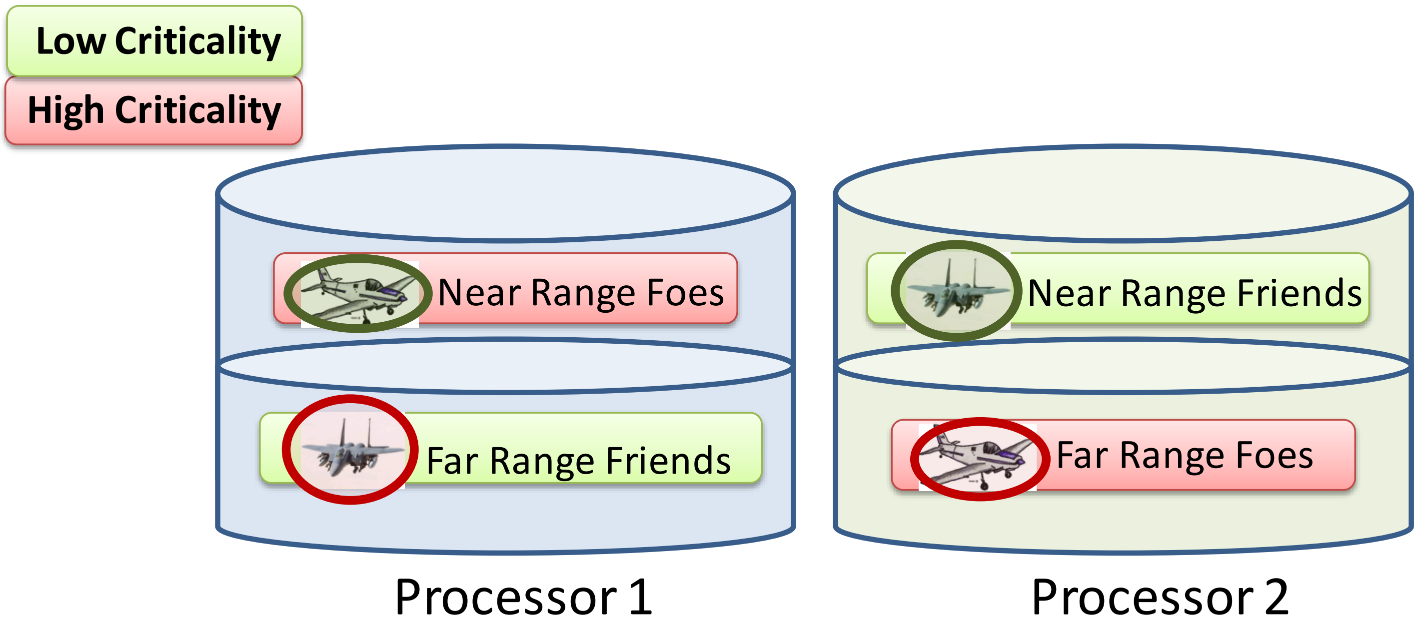


$$
\begin{array}{l c}
 & \begin{array}{cc} H & L \end{array} \\
\begin{array}{l} \text{Both overload} \\ \text{High criticality overload} \\ \text{Low criticality overload} \\ \text{No overload} \end{array} & \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}
\end{array}
$$

Assume that the system is schedulable without overloads

Under overloads only one task can meet its deadline

Criticality mixture strategy (T)



	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

T is better than S

Assume that the system
is schedulable without overloads
Under overloads only one
task can meet its deadline
*Assume that a uniprocessor
mixed-criticality scheduling algorithm
like ZSRM is used within each processor*

Generalization: Ductility Matrix

$$
\begin{array}{cc}
 & \xleftarrow{\quad\quad} \text{Criticality} \\
\text{Overload} \downarrow &
\begin{pmatrix}
d_{1,1} & d_{1,2} & d_{1,3} & \dots & d_{1,k} \\
d_{2,1} & d_{2,2} & d_{2,3} & \dots & d_{2,k} \\
\dots & & & \dots & \\
d_{2^k,1} & d_{2^k,2} & d_{2^k,3} & \dots & d_{2^k,k}
\end{pmatrix}
\end{array}
$$

Say we have ‘k’ criticality levels

2^k possible overload scenarios

- *All criticality levels overload* to *No overload*

Quantification of Ductility

$$P_d\,(\mathrm{D}) = \sum_{c=1}^{k} \left\{ \frac{1}{2^c} \frac{\sum_{r=1}^{2^k} d_{r,c}}{2^k} \right\}$$

*Scheme **S***

	H	L
Both overload	0	0
High criticality overload	0	1
Low criticality overload	1	0
No overload	1	1

*Scheme **T***

	H	L
Both overload	1	0
High criticality overload	1	0
Low criticality overload	1	0
No overload	1	1

For S, P_d (D) = 0.375

For T, P_d (D) = 0.5625

Shows that ***T*** is better than ***S***

Other Projection functions can be used

P_d(D) favors the more critical tasks **exponentially** over the lower criticality tasks

Outline

Mixed-criticality task scheduling problem

Zero-slack scheduling for uni-processors

- Zero-slack metrics & properties

Generalizing resource allocation to distributed mixed criticality tasks

- Generalized metric: Ductility matrix

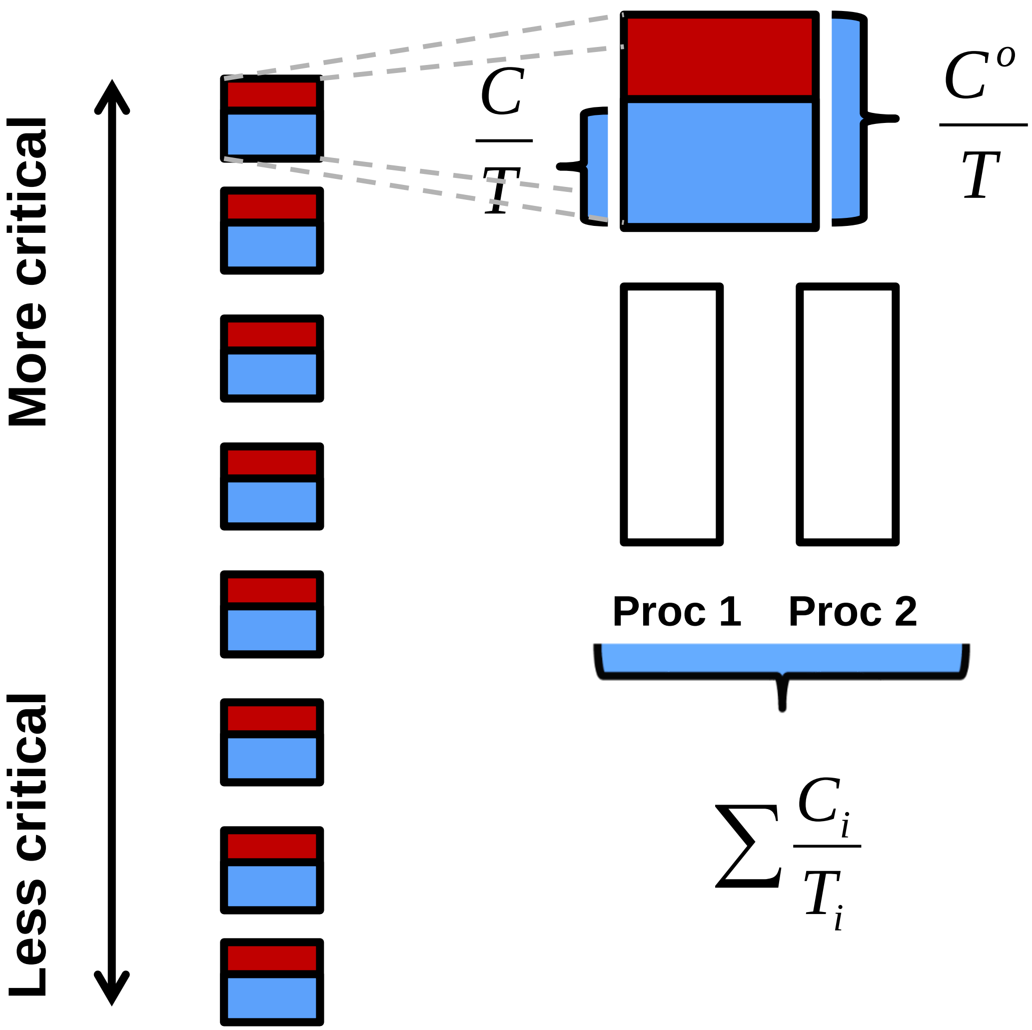
Compress-on-Overload Packing (COP)

- COP Performance

Radar surveillance case study

Conclusions

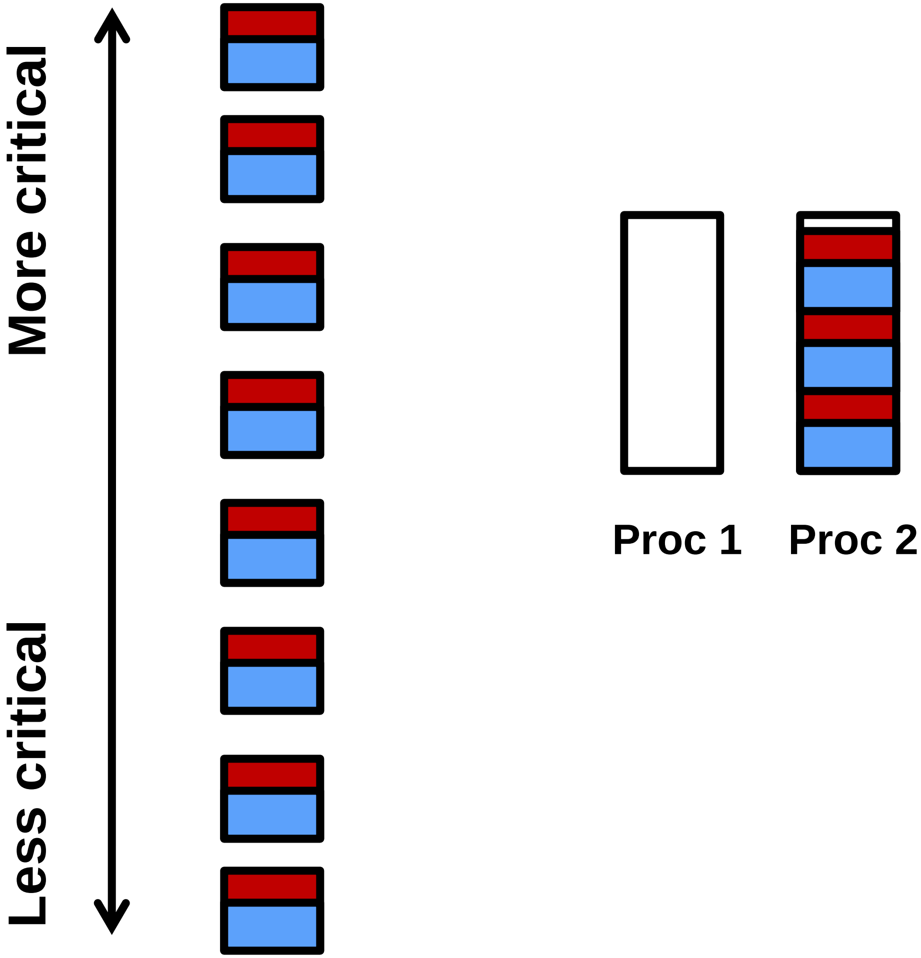
Compress-on-Overload Packing (COP)



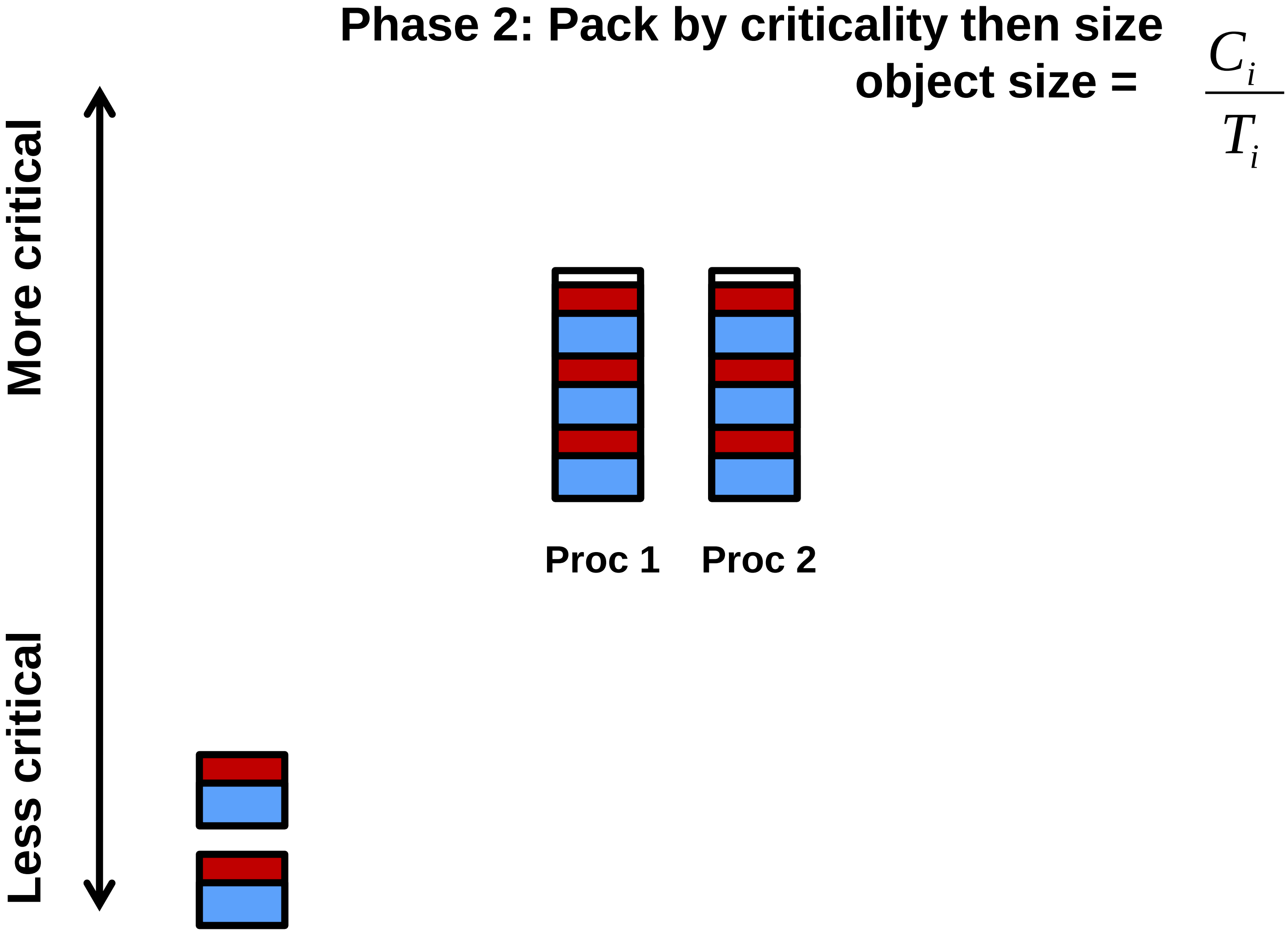
Compress-on-Overload Packing (COP)

Phase 1: Pack by criticality then size

object size = $\frac{C_i^{o}}{T_i}$



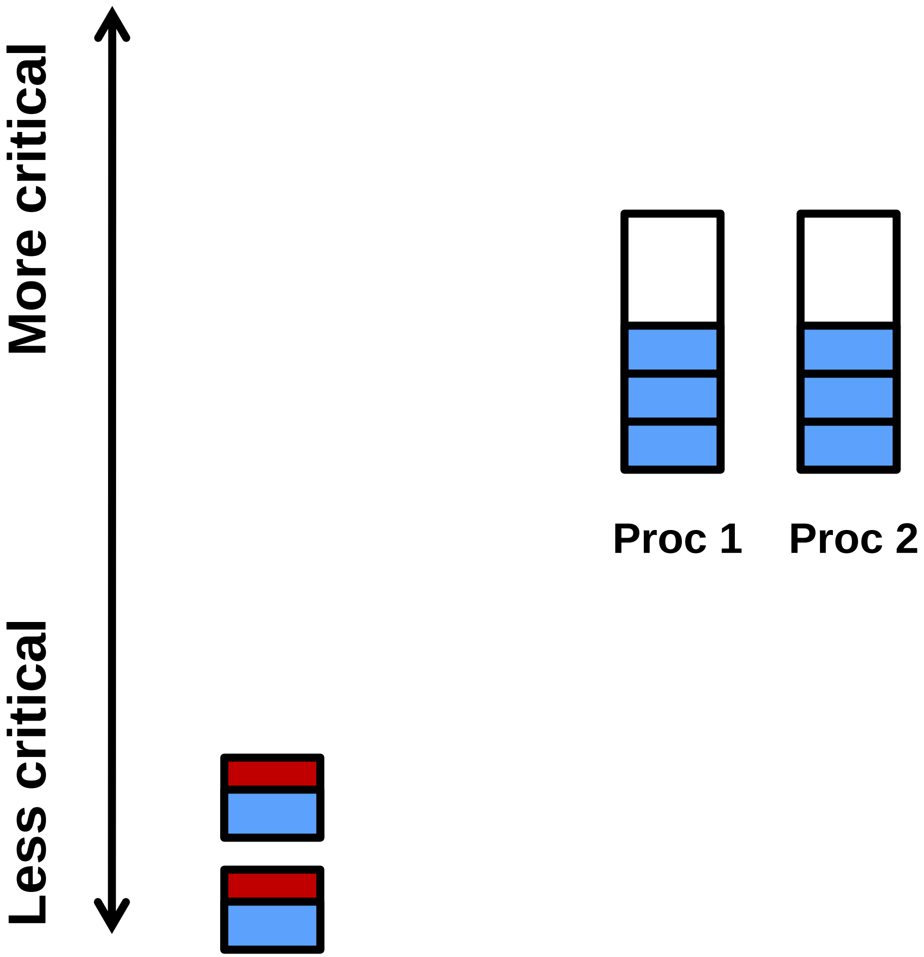
Compress-on-Overload Packing (COP)



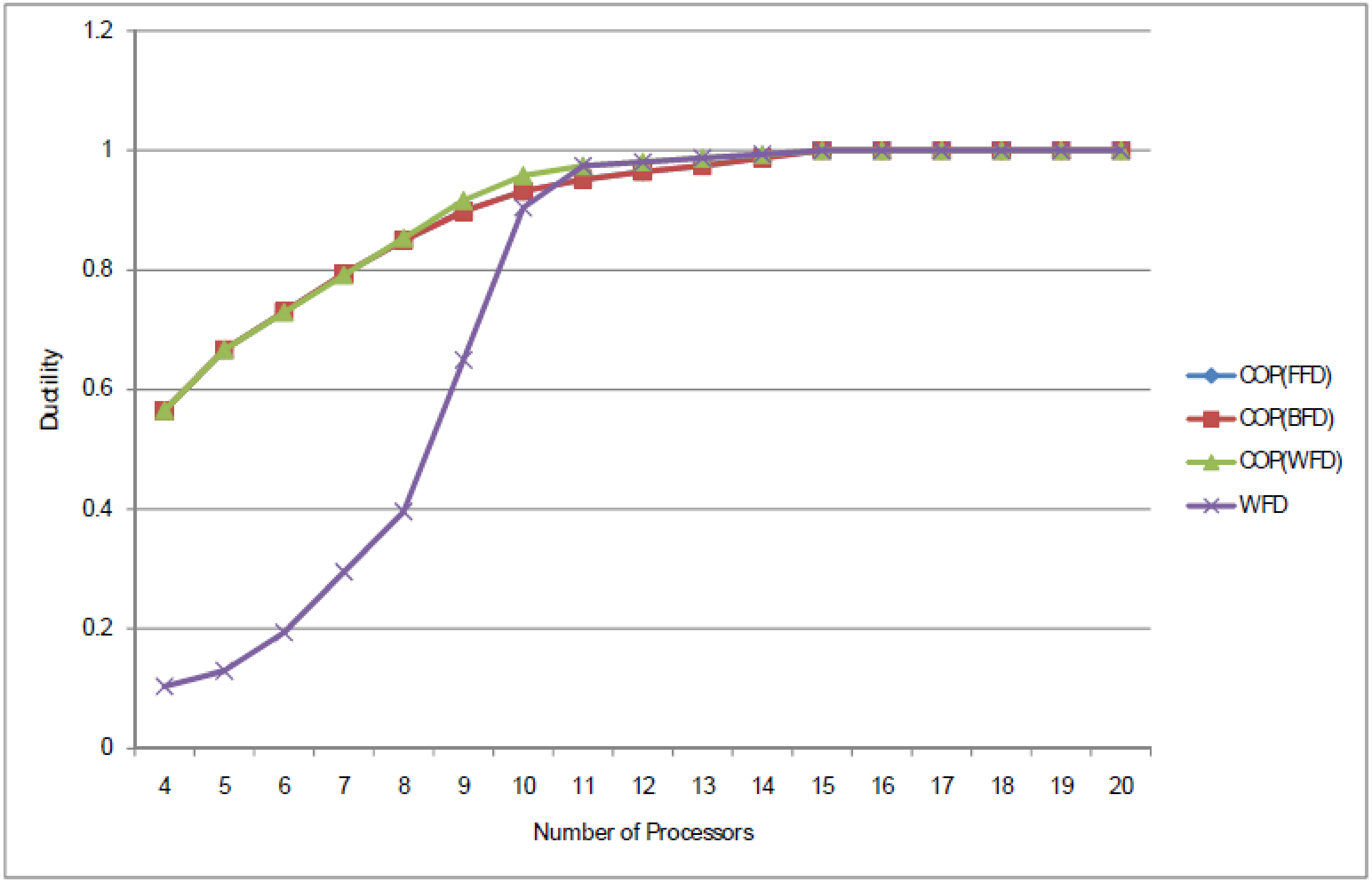
Compress-on-Overload Packing (COP)

Phase 2: Pack by criticality then size

object size = $\frac{C_i}{T_i}$

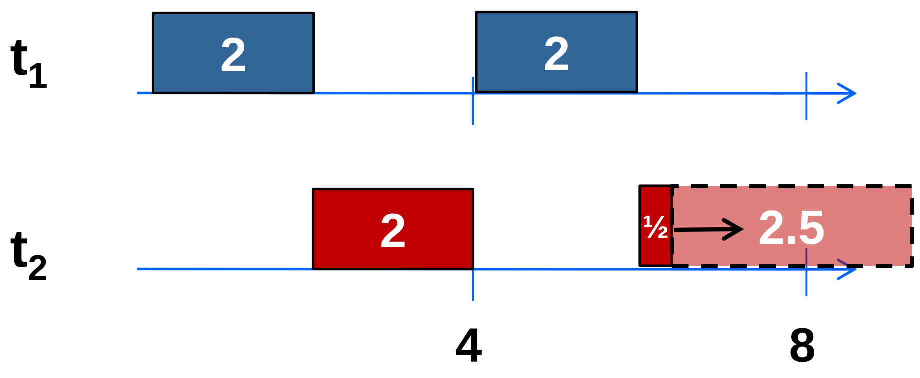


COP Performance



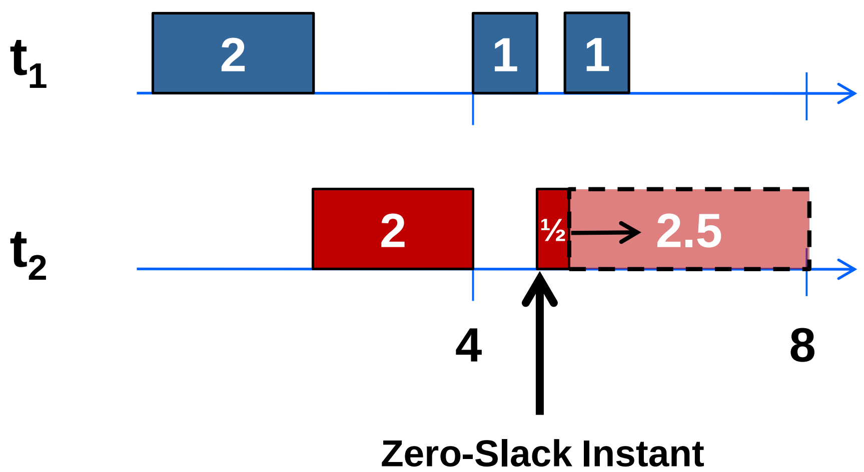
Overloading in Mixed-Criticality Systems

Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5



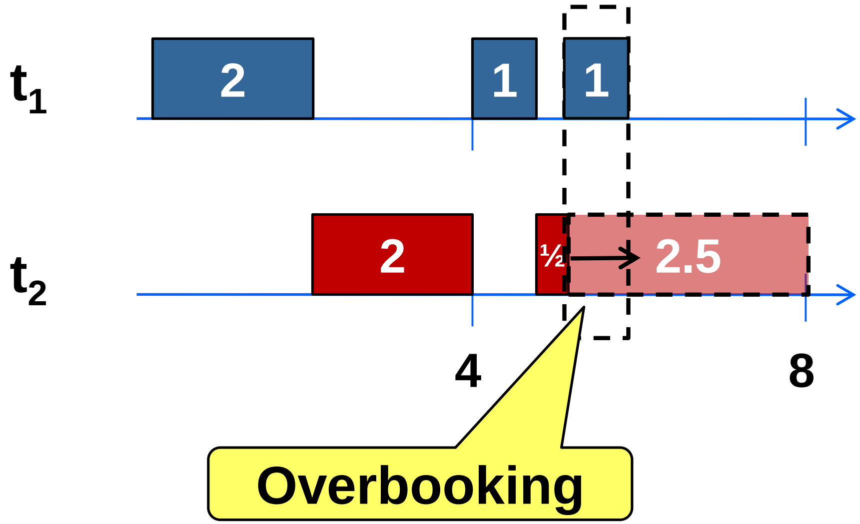
Zero-Slack Rate Monotonic

Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5



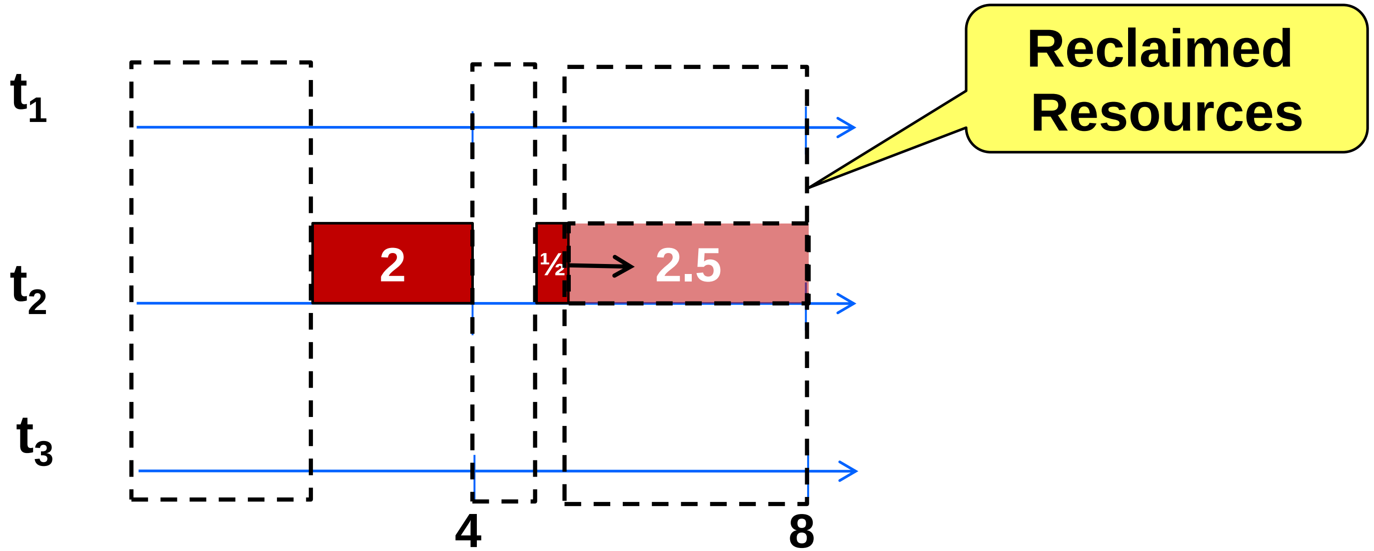
Zero-Slack Rate Monotonic

Task	Period	Criticality	WCET	NCET
t_1 Surveillance Cov.	4	Mission	2	2
t_2 Collision Avoid.	8	Safety	5	2.5



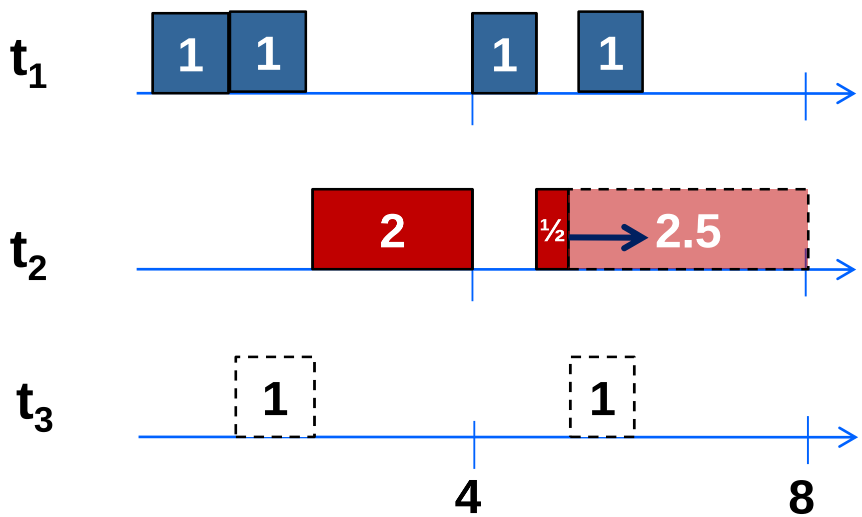
Reclaiming Resources in Mixed-Criticality Systems

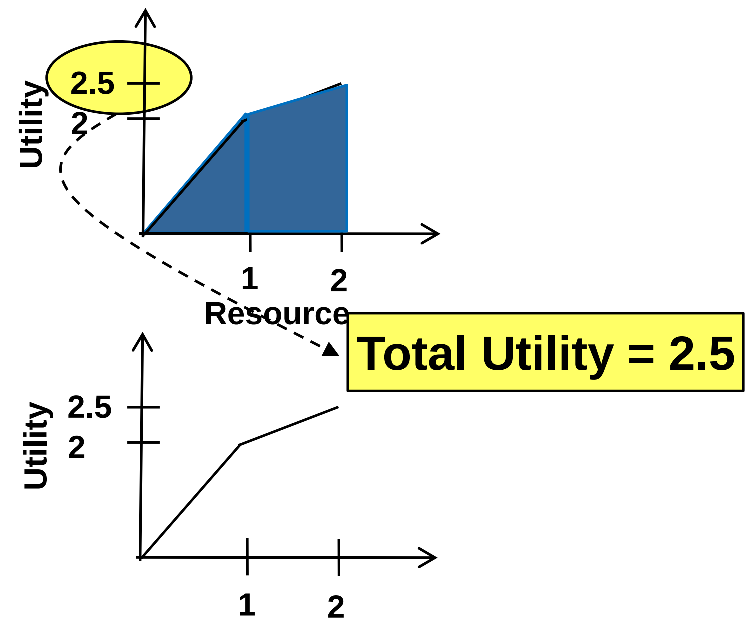
Task	Period	Criticality	WCET	NCET	Utility
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}



Using Reclaimed Resources to Maximized Utility

Task	Period	Criticality	WCET	NCET	Utility Levels
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}

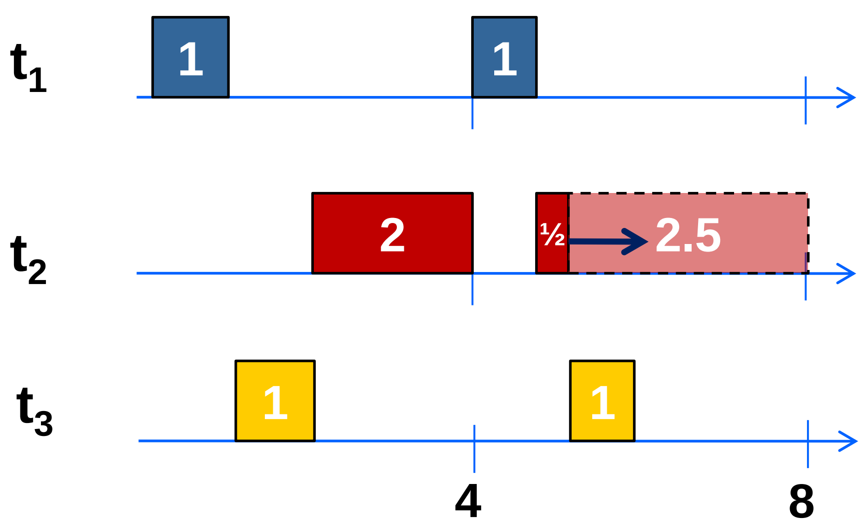


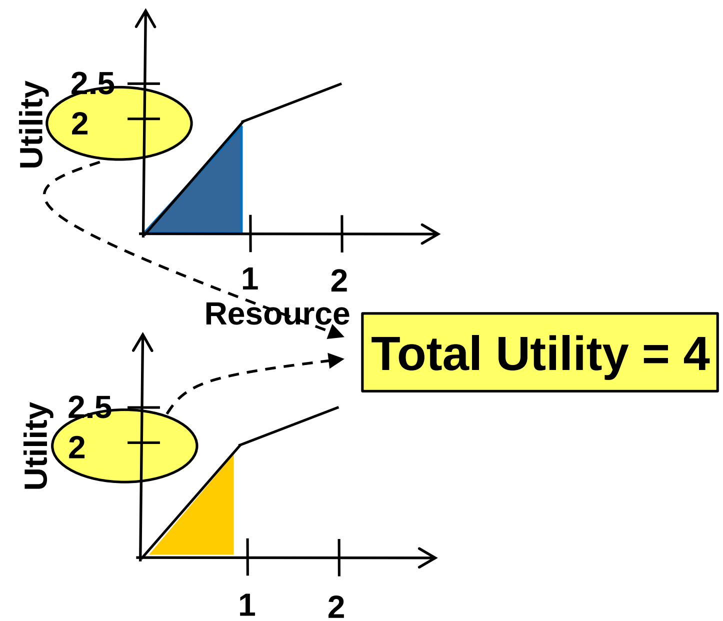


Utility Diminishes: Utility ≠ Criticality

Using Reclaimed Resources to Maximized Utility

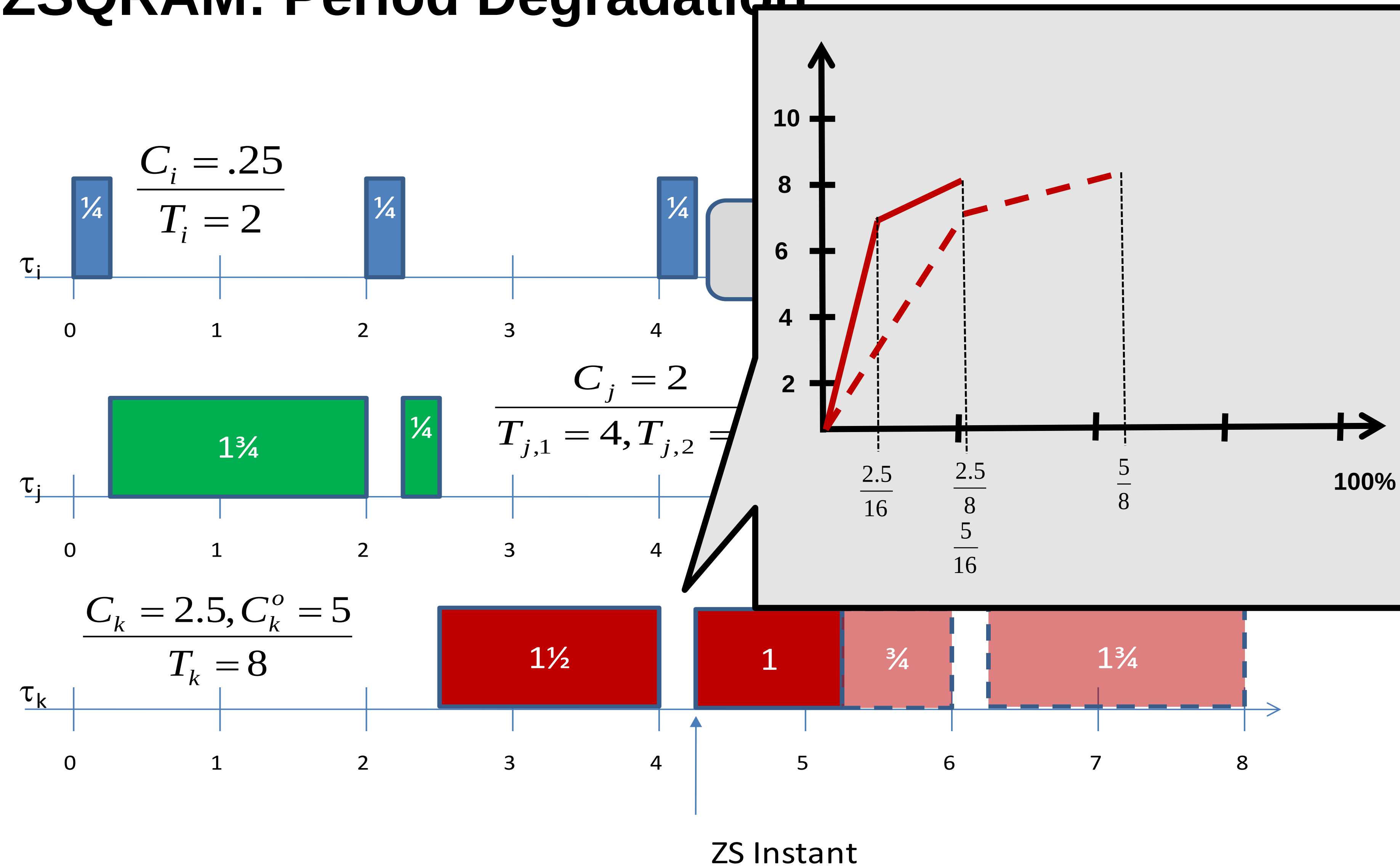
Task	Period	Criticality	WCET	NCET	Utility Levels
t_1 Surveillance Cov.	4	Mission	2	2	{2,2.5}
t_2 Collision Avoid.	8	Safety	5	2.5	
t_3 Amount of Intelligence	4	Mission	2	2	{2,2.5}





ZS-QRAM: More mission-critical utility from same resources

ZSQRAM: Period Degradation



ZSQRAM: Period Degradation